(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organizati n
International Bureau

(43) International Publication Date
21 June 2001 (21.06.2001)

PCT

(10) International Publication Number
WO 01/44927 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US00/41668

(22) International Filing Date: 27 October 2000 (27.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/434,715 5 November 1999 (05.11.1999) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road. Palo Alto. CA 94303 (US).

(72) Inventors: KOSCHE, Nicolai; 725 Fell Street #9, San Francisco. CA 94117 (US). DAMRON, Peter; 4430 Romilly Way, Fremont, CA 94536 (US).

(74) Agent: PARK, Richard; 508 2nd Street. Suite 201, Davis, CA 95616 (US).

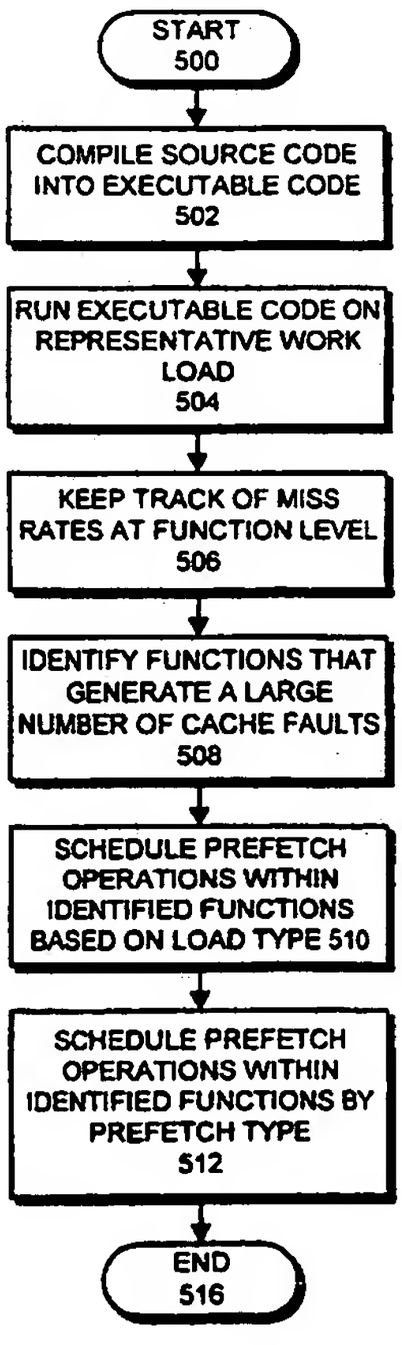
(81) Designated States *(national)*: AE, AG, AL, AM. AT, AU. AZ. BA. BB, BG, BR. BY, BZ. CA, CH, CN, CR. CU. CZ, DE. DK, DM. DZ. EE. ES. FI. GB, GD. GE. GH. GM, HR, HU, ID, IL. IN, IS, JP, KE, KG, KP, KR, KZ, LC. LK. LR. LS. LT. LU. LV, MA, MD. MG. MK. MN. MW. MX, MZ. NO. NZ, PL, PT. RO. RU, SD. SE. SG, SI. SK. SL, TJ, TM, TR. TT, TZ, UA. UG. UZ. VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE. LS. MW, MZ. SD. SL. SZ. TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ. TM), European patent (AT, BE, CH. CY, DE, DK, ES, FI. FR. GB, GR. IE. IT. LU, MC, NL, PT. SE), OAPI patent (BF. BJ. CF. CG, CI, CM, GA, GN, GW, ML. MR, NE. SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR PERFORMING PREFETCHING AT THE FUNCTION LEVEL



(57) Abstract: One embodiment of the present invention provides a system for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses. The system operates by compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor. Next, the system runs the executable code module in a training mode on a representative workload and keeps statistics on cache miss rates for functions within the executable code module. These statistics are used to identify a set of "hot" functions that generate a large number of cache misses. Next, explicit prefetch instructions are scheduled in advance of memory operations within the set of hot functions. In one embodiment, explicit prefetch operations are scheduled into the executable code module by activating prefetch generation at a start of an identified function, and by deactivating prefetch generation at a return from the identified function. In embodiment, the system further schedules prefetch operations for the memory operations by identifying a subset of memory operations of a particular type within the set of hot functions, and scheduling explicit prefetch operations for memory operations belonging to the subset.

WO 01/44927 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

METHOD AND APPARATUS FOR PERFORMING PREFETCHING AT THE FUNCTION LEVEL

Related Application

The subject matter of this application is related to the subject matter in a co-pending non-provisional application by the same inventors as the instant application and filed on the same day as the instant application entitled, "Method and Apparatus for Performing Prefetching at the Critical Section Level," having serial number TO BE ASSIGNED, and filing date TO BE ASSIGNED (Attorney Docket No. SUN-P4342-JTF).

BACKGROUND

Field of the Invention

The present invention relates to compilers for computer systems. More specifically, the present invention provides a method and an apparatus for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate a large number of cache misses.

Related Art

As processor clock speeds continue to increase at an exponential rate, memory latencies are becoming a major bottleneck to computer system performance. On some applications a processor can spend as much as half of its time waiting for outstanding memory operations to move data from cache or main memory into registers within the processor. A single memory operation can cause the processor to wait for many clock cycles if the memory operation causes a cache miss from fast L1 cache and a

corresponding access from slower L2 cache, or worse yet, causes a cache miss from L2 cache and a corresponding access from main memory.

It is possible to alleviate some of the performance limiting effects of memory operations by designing a system so that it can initiate a memory operation in advance of instructions that make use of the data returned from the memory operation. However, designing such capabilities into a processor can greatly increase the complexity of the processor. This increased complexity can increase the cost of the processor and can potentially decrease the clock speed of the processor if the additional complexity lengthens a critical path through the processor. Furthermore, the potential performance gains through the use of such techniques can be limited.

It is also possible to modify executable code during the compilation process so that it explicitly prefetches data associated with a memory operation in advance of where the memory operation takes place. This makes it likely that the data will be present in L1 cache when the memory operation occurs. This type of prefetching can be accomplished by scheduling an explicit prefetch operation into the code in advance of an associated memory operation in order to prefetch the data into L1 cache before the memory operation is encountered in the code.

Unfortunately, it is very hard to determine which data items should be prefetched and which ones should not. Prefetching all data items is wasteful because the memory system can become bottlenecked prefetching data items that are not referenced. On the other hand, analyzing individual memory operations to determine if they are good candidates for prefetching can consume a great deal of computational time.

What is needed is a method and an apparatus that selects a set of memory operations for prefetching without spending a great deal of time analyzing individual memory operations.

SUMMARY

One embodiment of the present invention provides a system for compiling source code into executable code that performs prefetching for memory operations

within regions of code that tend to generate cache misses. The system operates by compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor. Next, the system runs the executable code module in a training mode on a representative workload and keeps statistics on cache miss rates for functions within the executable code module. These statistics are used to identify a set of "hot" functions that generate a large number of cache misses. Next, explicit prefetch instructions are scheduled in advance of memory operations within the set of hot functions.

In one embodiment, explicit prefetch operations are scheduled into the executable code module by activating prefetch generation at a start of an identified function, and by deactivating prefetch generation at a return from the identified function.

In embodiment, the system further schedules prefetch operations for the memory operations by identifying a subset of memory operations of a particular type within the set of hot functions, and scheduling explicit prefetch operations for memory operations belonging to the subset. The particular type of memory operation can include, memory operations through pointers, memory operations involving static data, memory operations from locations that have not been previously accessed, memory operations outside of the system stack, and memory operations that are likely to be executed.

In one embodiment, the system schedules the prefetch operations by identifying a subset of prefetch operations with a particular property, and by scheduling the prefetch operations based on the property. For example, the particular property can include having an available issue slot, being located on an opposite side of a function call site from an associated memory operation, being located on the same side of a function call site from the associated memory operation, and being associated with a cache block that is not already subject to a scheduled prefetch operation.

One embodiment of the present invention provides a system for compiling source code into executable code that performs prefetching for memory operations

within critical sections of code that are subject to mutual exclusion. The system operates by compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor. Next, the system identifies a critical section within the executable code module by identifying a region of code between a mutual exclusion lock operation and a mutual exclusion unlock operation. The system schedules explicit prefetch instructions into the critical section in advance of associated memory operations.

In one embodiment, the system identifies the critical section of code by using a first macro to perform the mutual exclusion lock operation, wherein the first macro additionally activates prefetching. The system also uses a second macro to perform the mutual exclusion unlock operation, wherein the second macro additionally deactivates prefetching. Note that the second macro does not deactivate prefetching if the mutual exclusion unlock operation is nested within another critical section.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 illustrates a computer system in accordance with an embodiment of the present invention.

FIG. 2 illustrates load operations occurring within regions of executable code in accordance with an embodiment of the present invention.

FIG. 3A illustrates macros that enable and disable prefetching in accordance with an embodiment of the present invention.

FIG. 3B illustrates nesting of critical sections in accordance with an embodiment of the present invention.

FIG. 4 presents an example of prefetching loads that are likely to be executed accordance with an embodiment of the present invention.

FIG. 5 is a flow chart illustrating the process of creating code that prefetches loads within hot functions in accordance with an embodiment of the present invention.

FIG. 6 is a flow chart illustrating the process of creating code that prefetches loads within critical sections in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

The data structures and code described in this detailed description are typically stored on a computer readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Computer System

FIG. 1 illustrates the internal structure of computer system 100 in accordance with an embodiment of the present invention. In particular, FIG. 1 illustrates the memory hierarchy for computer system 100, which includes registers 104 within central processing unit (CPU) 102, L1 cache 106, prefetch cache 108, L2 cache 110, memory 112 and storage device 116.

CPU 102 can include any type of processing engine that can be used in a computer system, including, but not limited to, a microprocessor, a mainframe

processor, a device controller, a processor within a personal organizer and processing circuitry within an appliance. Registers 104 are internal registers within CPU 102 into which data is loaded from L1 cache 106, prefetch cache 108, L2 cache 110 or memory 112. Once data is loaded into registers 104, CPU 102 can perform computational operations on the data. (Although this disclosure often discusses prefetching for "load" operations, please note that the discussion applies to any memory operations that can benefit from prefetching, including stores and other memory references.)

Data is loaded into registers 104 from L1 cache 106. L1 cache 106 is a high-speed cache memory of limited size that is located in close proximity to CPU 102. In some embodiments, L1 cache 106 may be located within the same semiconductor chip as CPU 102.

Similarly, data is loaded into registers 104 from prefetch cache 108. Prefetch cache 108 is also a high-speed cache memory of limited size that is located in close proximity to CPU 102. The difference between prefetch cache 108 and L1 cache 106 is that prefetch cache 108 holds data that is explicitly prefetched, whereas L1 cache 106 holds data that has been recently referenced, but not prefetched. The use of prefetch cache 108 allows speculative prefetching to take place without polluting L1 cache 106.

Data is loaded into L1 cache 106 and prefetch cache 108 from L2 cache 110. L2 cache 110 is considerably larger that L1 cache 106 or prefetch cache 108. However, L2 cache is located farther from CPU 102, and hence accesses to L2 cache 110 take more time than accesses to L1 cache 106 or prefetch cache 108. However, note that accesses to L2 cache take less time than accesses to memory 112.

L1 cache 106, prefetch cache 108 and L2 cache 110 may be designed in a number of ways. For example, they may include direct-mapped caches, fully associative caches or set-associative caches. They may also include write-through or write-back caches.

Data is loaded into L2 cache from memory 112. Memory 112 can include any type of random access memory that can be used to store code and/or data for use by CPU 102. In the embodiment of the present invention illustrated in FIG. 1, memory

112 contains code with explicit prefetch instructions that are inserted at the function level or at the critical section level as is discussed below with reference to FIGs. 2-6.

Data is loaded into memory 112 from files within storage device 116. Storage device 116 can include any type of non-volatile storage device for storing code and/or data to be operated on by CPU 102. In one embodiment, storage device 116 includes a magnetic disk drive.

FIG. 1 also illustrates how CPU 102 can be coupled to server 122 through network 120. Network 120 can include any type of wire or wireless communication channel capable of coupling together computing nodes. This includes, but is not limited to, a local area network, a wide area network, or a combination of networks. In one embodiment of the present invention, network 120 includes the Internet. Server 122 can include any computational node including a mechanism for servicing requests from a client for computational or data storage resources. In embodiment of the present invention, server 122 is a file server that contains executable code to by executed by CPU 102. Also note that although network 120 is illustrated as being directly coupled to CPU 102, in general network 102 can be coupled to other locations within the computer system illustrated in FIG. 1.

Note that FIG. 1 does not illustrate the many possible ways in which components of the memory hierarchy can be coupled together through various data paths and busses. Also note that the present invention can generally be applied to any type of computer system with prefetch capability, not just the specific computer system illustrated in FIG. 1.

Loads within Regions of Code

FIG. 2 illustrates load operations occurring within regions of executable code in accordance with an embodiment of the present invention. FIG. 2 illustrates a section of code that is divided into regions, including region A 202, region B 204 and region C 206. These regions include load operations to load data from the memory hierarchy into registers 104 within CPU 102. These load operations are illustrated in the middle column of FIG. 1. Note that the section of code also includes many

intervening non-load operations, which are not illustrated. These non-load operations manipulate the data that is pulled into registers 104 by the load operations.

The right-hand column of FIG. 2 illustrates the results of the load operations. More specifically, the first two load operations from the top of FIG. 2 (which are within region A 202) are retrieved from L1 cache 106. The next four load operations (within region B 204) are retrieved from L2 cache 110, memory 112, L2 cache 110 and L2 cache 110, respectively. The last two loads (within region C 206) are retrieved from L1 cache 106.

In this example, all of the loads within region B 204 generate cache misses from L1 cache 106 to L2 cache 110. One of these loads generates an additional cache miss in L2 cache 110 and a corresponding access to memory 112. Region B 204 is referred to as a "hot" region because a high percentage of the loads within region B 204 generate cache misses. Hence, the loads within region B 204 are good candidates for prefetching.

Note that region boundaries can be determined in a number of ways. In one embodiment of the present invention, region boundaries are function boundaries. In another embodiment, region boundaries are critical section boundaries. Note that loads within critical sections tend to generate a large number of cache misses because critical sections typically access shared data, which is prone to cache misses. Region boundaries may also encompass arbitrary "hot" regions of code that are specified by a user. Regions boundaries can also encompass complete source files, which can be specified in a command line.

Prefetching for Critical Sections

FIG. 3A illustrates mutual exclusion macros that enable and disable prefetching in accordance with an embodiment of the present invention. The first macro at the top of FIG. 3A is a mutual exclusion lock macro that turns on a prefetching feature of the compiler with specific prefetch properties before locking a mutual exclusion variable. This prefetching feature attempts to perform prefetching for all load operations unless the prefetch operations are filtered out as is discussed

below with reference to FIGs. 5 and 6. Note that the mutual exclusion variable can generally include any type of mutual exclusion variable, such as a mutual exclusion variable associated with a spin lock, a semaphore, a read-writer lock, a turnstile, a mutex lock, an adaptive mutex lock, or any other mutual exclusion mechanism.

Also note that the prefetching feature can have specific prefetch properties for associated load and prefetch instructions. These properties are discussed in more detail below. Hence, different mutual exclusion macros can activate different prefetching properties. In other embodiment of the present invention different prefetching properties can be activated at the function level, the file level or within an arbitrary region of code. These different prefetching properties can be activated and deactivated by different regions markers (such as mutual exclusion macros) that are specific to particular properties. Note that these different region markers can be nested.

The second macro in FIG. 3A illustrates a corresponding mutual exclusion unlock macro that unlocks the mutual exclusion variable and turns off the prefetching feature. In one embodiment of the present invention, the system checks for an unmatched second macro that deactivates prefetching and is not preceded by a matching first macro that activates prefetching. If such an unmatched second macro is encountered, the may system signal an error condition.

FIG. 3B illustrates nesting of critical sections in accordance with an embodiment of the present invention. In many applications, critical sections are nested. For example, in FIG. 3B, critical section B 304, which is bounded by a mutex_lock(B) and mutex_unlock(B), is nested within critical section A 302, which is bounded by a mutex_lock(A) and mutex_unlock(A). In this case, the turnoff_prefetch() function keeps track of the number of nested critical sections and does not turn off prefetching at the end of a nested critical section. For example, the mutex_unlock(B) call within FIG. 3B does not turn off prefetching because it is associated with nested critical section B 304. However, the mutex_unlock(A) call does turn off prefetching because subsequent code is outside of any critical section and is not subject to prefetching.

FIG. 6 is a flow chart illustrating the process of creating code that prefetches loads within critical sections in accordance with an embodiment of the present invention. The system starts by compiling a source code module into executable code instructions to produce a corresponding executable code module (step 602). In doing so, the system identifies critical sections (step 604). This can be done by using the mutex_lock() and mutex_unlock() macros illustrated in FIG. 3A. Alternatively, the compiler can be modified to look for mutual exclusion lock and unlock operations in order to enable and disable prefetching.

Next, the system examines the load operations within the critical sections and schedules prefetch operations for certain types of load operations (step 606). This can greatly reduce the number of prefetch operations. For example, the system can choose to prefetch, loads through pointers, loads of static data, loads through pointer and loads of static data, loads from outside the system stack, or loads that are likely to be executed. Note that loads that are likely to be executed can be identified by running the executable code in a training mode. Also note that loads within the system stack or loads from locations that have been previously loaded are unlikely to generate cache misses and are hence bad candidates for prefetching.

The system can also schedule prefetch operations that appear within critical sections based upon properties of the prefetch operations (step 608). For example, the system can choose to schedule a prefetch operations only if there exists an available load issue slot and available outstanding loads for the prefetch operation. Note that a typical load store unit in a processor has a small number of load issue slots available as well as a limited number of outstanding loads. If these load issue slots are filled, it makes little sense to schedule a prefetch because no load issue slots are available for the prefetch. The system can also schedule a prefetch operation on an opposite side of a function call site from an associated load operation (or alternatively on the same side of the function call site). This can be useful if the call site is for a function that is unlikely to affect the cache, such as a mutex lock function. For other types of functions it makes little sense to issue a prefetch before the function call, because the function call is likely to move the flow of execution to another region of the code for a

long period of time. The system can also schedule a prefetch for a cache block that is not already subject to a scheduled prefetch operation.

At this point, the source code is ready for normal program execution.

Prefetching Loads That Are Likely To Be Executed

FIG. 4 presents an example of prefetching loads that are likely to be executed in accordance with an embodiment of the present invention. Function 400 is divided into four basic blocks 402-405. A basic block is a section of code that executes without a change in control flow. Hence, a basic block contains at most one branch or function call at the end of the block. In FIG. 4, there is a conditional branch at the end of basic block 402, which goes to either basic block 404 or basic block 403. Later on. these separate branch paths rejoin in basic block 405.

Each of the illustrated basic blocks 402-405 includes load operations. More specifically, basic block 402 includes loads A and B. Basic block 403 includes loads D and E. Basic block 404 includes loads F and G. Finally, basic block 405 includes loads H, I, J and K.

In the example illustrated in FIG. 4, assume that function 400 is a "hot" function that has exhibited a large number of cache misses while running on a representative workload. In this example, the system starts by filtering out loads that are directed to the system stack, because these loads are unlikely to generate cache misses. This eliminates loads C, G and H.

Next, the system eliminates loads that are not likely to be executed. Assume that basic blocks 402, 404 and 405 contain likely executed load operations. This eliminates loads D and E. Note that the system can identify the load instructions that are likely to be executed by running a program containing function 400 in a "training mode" on a representative workload and by keeping statistics on which instructions are executed through function 400.

Next, the system schedules prefetches up the likely execution path. In doing so, the system ensures that the number of outstanding prefetches does not exceed the number of available load issue slots in the system's load store unit and the maximum

number of outstanding loads. The example illustrated in FIG. 4 assumes there are four outstanding loads available. Hence, at the beginning of basic block 402, the system prefetches loads B, F and I prior to load A. (Note that the three prefetches for B, F and I plus the load of A will take up the four load issue slots). Next, assuming that the prefetch of B completes immediately after the load of A completes, another outstanding load becomes available and the system prefetches load J. Later on, assuming the prefetch of F completes before load F is encountered, the system prefetches load K.

Note that the technique of prefetching loads that are likely to be executed can be performed for any region of code, and is not limited to a function. For example, the system can also prefetch loads that are likely to be executed within a critical section, or any other arbitrary section of code.

Prefetching for Selected Functions

FIG. 5 is a flow chart illustrating the process of creating code that prefetches loads within hot functions in accordance with an embodiment of the present invention. The system starts by compiling a source code module into executable code instructions to produce a corresponding executable code module (step 502).

Next, the system determines which functions within the executable module tend to create a large number of cache misses. We refer to these functions as "hot functions." The system does so by running the executable module in a training mode on a representative workload (step 504), and by keeping statistics on cache miss rates at the function level (step 506). Next, the system uses these statistics to identify functions that tend to generate a large number of cache misses (step 508).

Next, the system examines all load operations within the hot functions and schedules prefetch operations for certain types of load operations (as was done above for critical sections) (step 510). The system can also schedule prefetch operations that appear within hot functions based upon properties of the prefetch operations (step 512). At this point, the source code is ready for normal program execution.

The foregoing descriptions of embodiments of the invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the invention. The scope of the invention is defined by the appended claims.

What Is Claimed Is:

1. A method for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses, comprising:

compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor;

identifying functions containing memory operations that tend to generate a large number of cache misses by;

running the executable code module on the processor in a training mode on a representative workload,

keeping statistics on cache miss rates for memory operations within functions within the executable code module, and

identifying a set of functions that generate a large number of cache misses; and

scheduling explicit prefetch instructions into the executable code module in advance of memory operations within the identified set of functions, so that prefetch operations are performed for memory operations within the set of functions that generate the large number of cache misses.

2. The method of claim 1, wherein scheduling explicit prefetch operations into the executable code module includes,

activating prefetch generation at a start of an identified function; and

deactivating prefetch generation at a return from the identified function.

3. The method of claim 2, wherein activating prefetch generation includes activating prefetch generation in response to prefetch generation being specified in a command line.

4. The method of claim 1, further comprising:

identifying a critical section within the executable code module by identifying a region of code between a mutual exclusion lock operation and a mutual exclusion unlock operation; and

scheduling explicit prefetch instructions into the executable code module in advance of memory operations located within the critical section, so that prefetch operations are performed for memory operations within the critical section.

5. The method of claim 1, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of memory operations of a particular type within the identified set of functions; and

scheduling explicit prefetch operations for memory operations belonging to the subset.

6. The method of claim 5, wherein the particular type of memory operation includes one of,

memory operations through pointers;

memory operations involving static data;

memory operations from locations that have not been previously accessed;

memory operations outside a system stack; and

memory operations that are likely to be executed.

7. The method of claim 1, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of prefetch operations with a particular property that are associated with memory operations within the identified set of functions; and

scheduling explicit prefetch operations for prefetch operations belonging to the subset based on properties of the subset.

8. The method of claim 7, wherein the particular property of the subset of prefetch operations includes, but is not limited to, one of,

existence of an available issue slot for the prefetch operation;

being located on the same side of a function call site from an associated memory operation;

being located on an opposite side of a function call site from an associated memory operation; and

being associated with a cache block that is not already subject to a scheduled prefetch operation.

9. A computer readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses, comprising:

compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor;

identifying functions containing memory operations that tend to generate a large number of cache misses by,

running the executable code module on the processor in a training mode on a representative workload,

keeping statistics on cache miss rates for memory operations within functions within the executable code module, and

identifying a set of functions that generate the large number of cache misses; and

scheduling explicit prefetch instructions into the executable code module in advance of memory operations within the identified set of functions, so that prefetch operations are performed for memory operations within the set of functions that generate the large number of cache misses.

10. The computer-readable storage medium of claim 9, wherein scheduling explicit prefetch operations into the executable code module includes,

activating prefetch generation at a start of an identified function; and

deactivating prefetch generation at a return from the identified function.

11. The computer-readable storage medium of claim 10, wherein activating prefetch generation includes activating prefetch generation in response to prefetch generation being specified in a command line.

12. The computer-readable storage medium of claim 9, wherein the method embodied within the instructions stored within the computer-readable storage medium further comprises:

identifying a critical section within the executable code module by identifying a region of code between a mutual exclusion lock operation and a mutual exclusion unlock operation; and

scheduling explicit prefetch instructions into the executable code module in advance of memory operations located within the critical section, so that prefetch operations are performed for memory operations within the critical section.

13. The computer-readable storage medium of claim 9, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of memory operations of a particular type within the identified set of functions; and

scheduling explicit prefetch operations for memory operations belonging to the subset.

14. The computer-readable storage medium of claim 13, wherein the particular type of memory operation includes, but is not limited to, one of,

memory operations through pointers;

memory operations involving static data;

memory operations from locations that have not been previously accessed;

memory operations outside a system stack; and

memory operations that are likely to be executed.

15. The computer-readable storage medium of claim 9, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of prefetch operations with a particular property that are associated with memory operations within the identified set of functions; and

scheduling explicit prefetch operations for prefetch operations belonging to the subset based on properties of the subset.

16. The computer-readable storage medium of claim 15, wherein the particular property of the subset of prefetch operations includes, but is not limited to, one of,

existence of an available issue slot for the prefetch operation;

being located on the same side of a function call site from an associated memory operation;

being located on an opposite side of a function call site from an associated memory operation; and

being associated with a cache block that is not already subject to a scheduled prefetch operation.

17. An apparatus that compiles source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses, comprising:

a compiling mechanism that compiles a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor;

an identification mechanism that identifies functions containing memory operations that tend to generate a large number of cache misses, the identification mechanism being configured to,

run the executable code module on the processor in a training mode on a representative workload,

keep statistics on cache miss rates for memory operations within functions within the executable code module, and

identify a set of functions that generate the large number of cache misses; and

a scheduling mechanism that schedules explicit prefetch instructions into the executable code module in advance of memory operations within the identified set of functions, so that prefetch operations are performed for memory operations within the set of functions that generate the large number of cache misses.

18. The apparatus of claim 17, wherein the scheduling mechanism is configured to,

activate prefetch generation at a start of an identified function; and

deactivate prefetch generation at a return from the identified function.

19. The apparatus of claim 18, wherein the scheduling mechanism is configured to activate prefetch generation in response to prefetch generation being specified in a command line.

20. The apparatus of claim 17, wherein:

the identification mechanism is further configured to identify a critical section within the executable code module by identifying a region of code between a mutual exclusion lock operation and a mutual exclusion unlock operation; and

the scheduling mechanism is further configured to schedule explicit prefetch instructions into the executable code module in advance of memory operations located

within the critical section, so that prefetch operations are performed for memory operations within the critical section.

21. The apparatus of claim 17, wherein the scheduling mechanism is further configured to:

identify a subset of memory operations of a particular type within the identified set of functions; and to

schedule explicit prefetch operations for memory operations belonging to the subset.

22. The apparatus of claim 21, wherein the particular type of memory operation includes, but is not limited to, one of,

memory operations through pointers;

memory operations involving static data;

memory operations from locations that have not been previously accessed;

memory operations outside a system stack; and

memory operations that are likely to be executed.

23. The apparatus of claim 17, wherein the scheduling mechanism is further configured to:

identify a subset of prefetch operations of with a particular property that are associated with memory operations within the identified set of functions; and to

schedule explicit prefetch operations for prefetch operations belonging to the subset based on properties of the subset.

24. The apparatus of claim 23, wherein the particular property of the subset of prefetch operations includes, but is not limited to, one of,

existence of an available issue slot for the prefetch operation;

being located on the same side of a function call site from an associated memory operation;

being located on an opposite side of a function call site from an associated memory operation; and

being associated with a cache block that is not already subject to a scheduled prefetch operation.

25. A method for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses, comprising:

compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor;

identifying a region of code containing memory operations that tend to generate a large number of cache misses; and

scheduling explicit prefetch instructions into the executable code module in advance of memory operations within the identified a region of code, so that prefetch operations are performed for memory operations within the region of code that tends to generate the large number of cache misses.

26. The method of claim 25, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of memory operations of a particular type within the identified set of functions; and

scheduling explicit prefetch operations for memory operations belonging to the subset.

27. The method of claim 26, wherein the particular type of memory operation includes, but is not limited to, one of,

memory operations through pointers;

memory operations involving static data;

memory operations from locations that have not been previously accessed;

memory operations outside a system stack; and

memory operations that are likely to be executed.

28. The method of claim 26, wherein the particular type of memory operation is specified by a region marker for the region of code.

29. The method of claim 25, wherein scheduling explicit prefetch instructions into the executable code module further comprises:

identifying a subset of prefetch operations with a particular property that are associated with memory operations within the identified set of functions; and

scheduling explicit prefetch operations for prefetch operations belonging to the subset based on properties of the subset.

30. The method of claim 29, wherein the particular property of the subset of prefetch operations includes, but is not limited to, one of,

existence of an available issue slot for the prefetch operation;

being located on the same side of a function call site from an associated memory operation;

being located on an opposite side of a function call site from an associated memory operation; and

being associated with a cache block that is not already subject to a scheduled prefetch operation.

31. The method of claim 29, wherein the particular property of the subset of prefetch operations is specified by a region marker for the region of code.

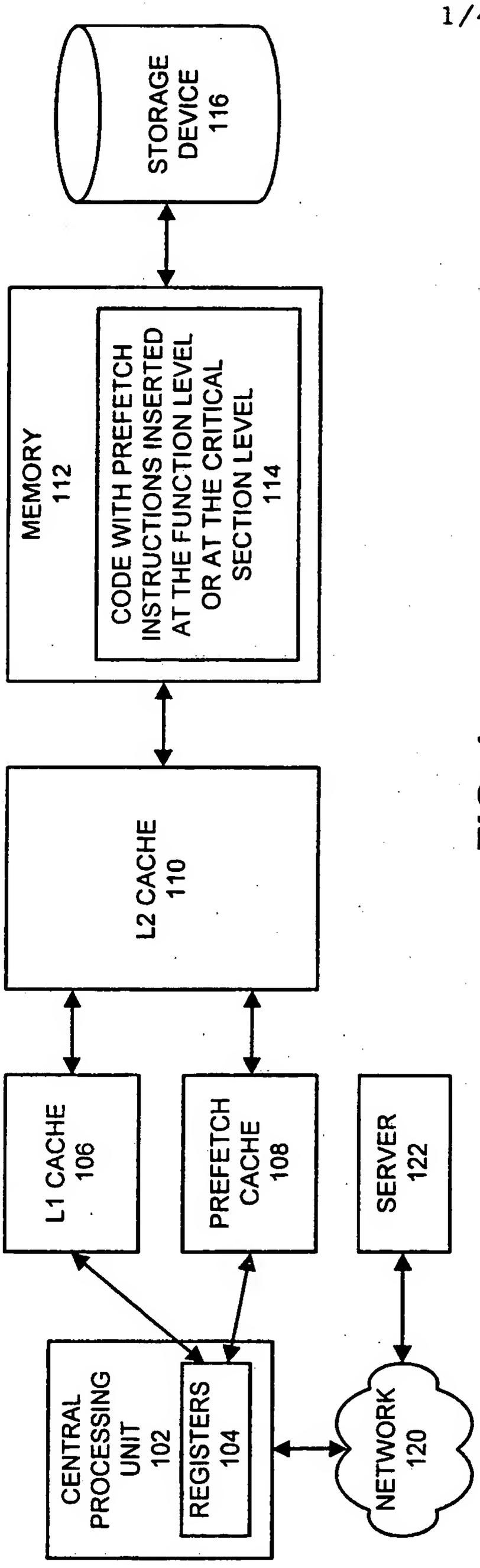

STORAGE DEVICE 116
MEMORY 112
CODE WITH PREFETCH INSTRUCTIONS INSERTED AT THE FUNCTION LEVEL OR AT THE CRITICAL SECTION LEVEL 114
L2 CACHE 110
L1 CACHE 106
PREFETCH CACHE 108
SERVER 122
CENTRAL PROCESSING UNIT 102
REGISTERS 104
NETWORK 120

FIG. 1

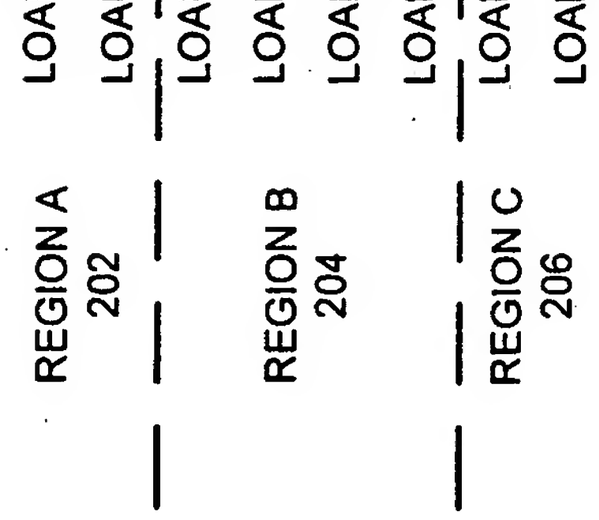

REGION A 202
REGION B 204
REGION C 206
LOAD
L1
L2
MEMORY
LARGE NUMBER OF CACHE FAULTS (HOT REGION)

FIG. 2

```
# define    mutex_lock(lock)        \
            ( _ _turnon_prefetch()  \
            mutex_lock(lock) )

# define    mutex_unlock(lock)      \
            ( mutex_unlock(lock)    \
            _ _turnoff_prefetch() )
```

FIG. 3A

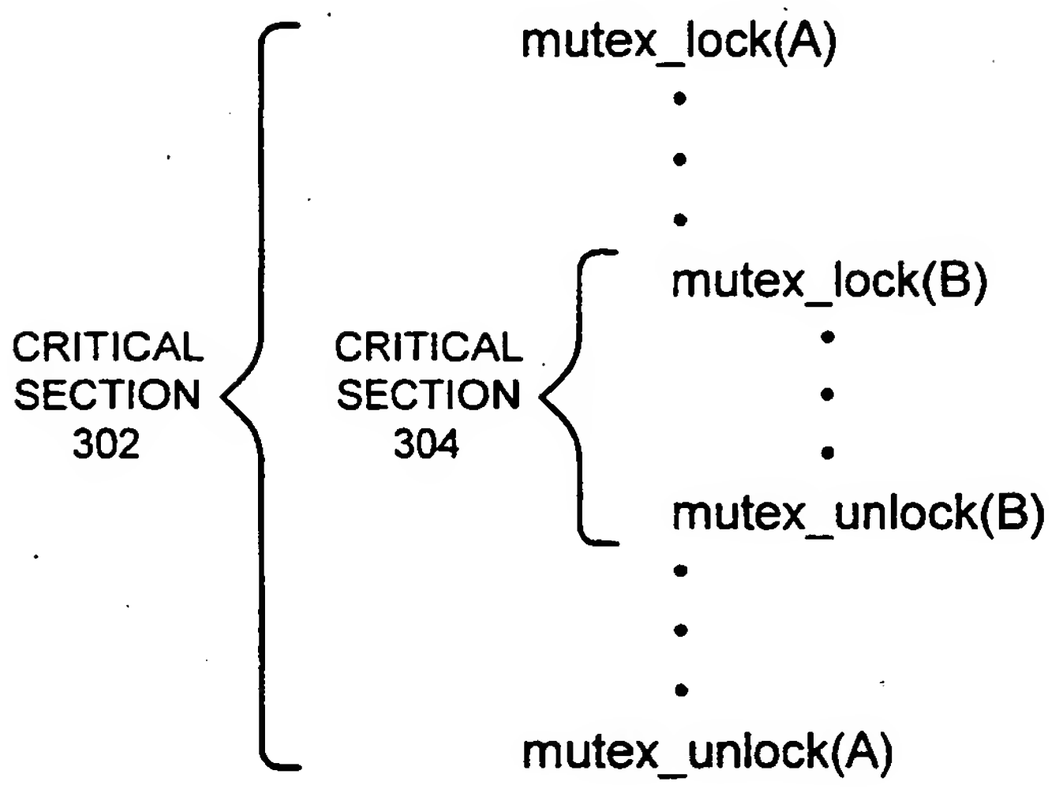


FIG. 3B

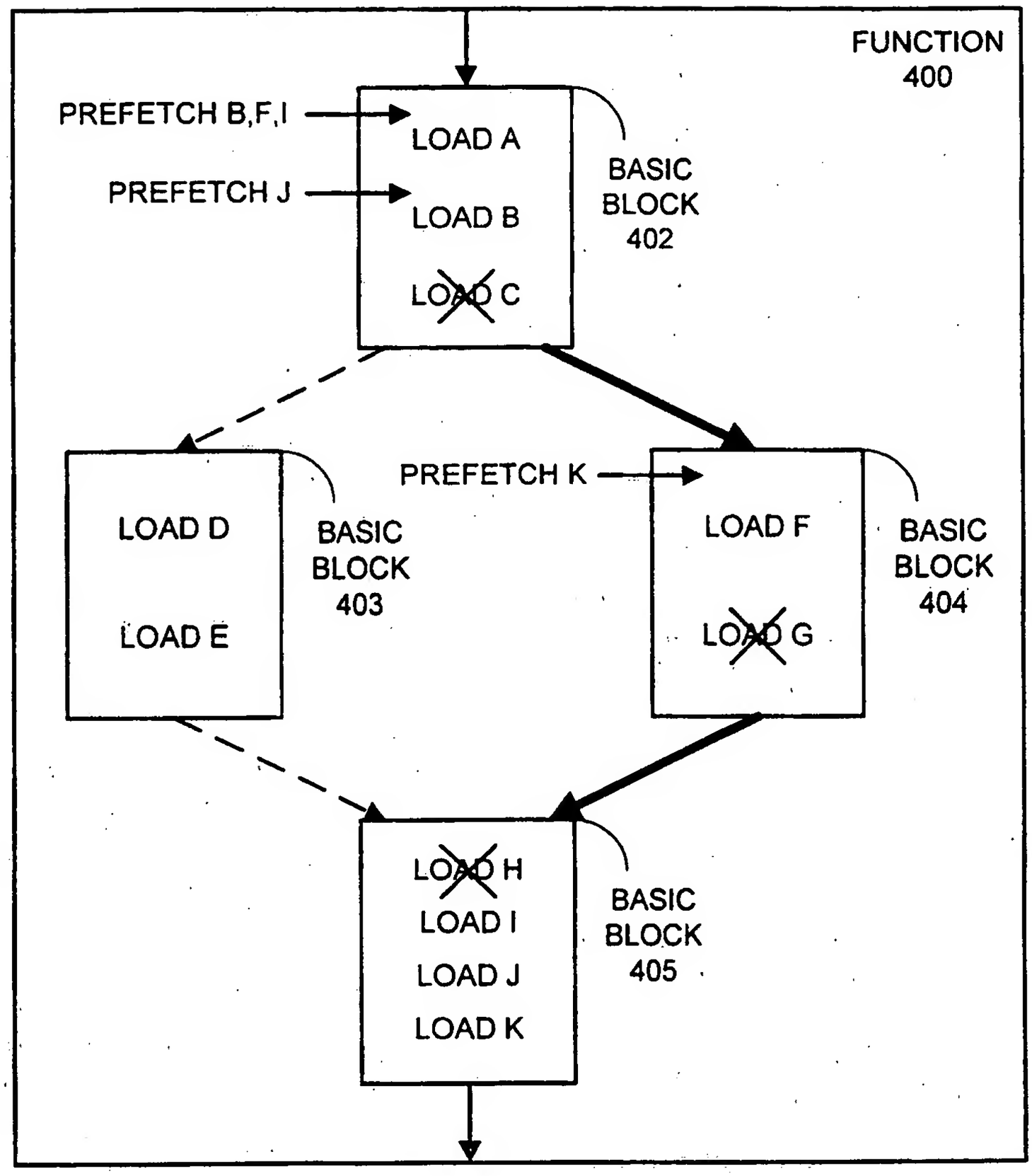

FUNCTION
400
PREFETCH B,F,I
LOAD A
PREFETCH J
LOAD B
LOAD C
BASIC
BLOCK
402
PREFETCH K
LOAD D
LOAD E
BASIC
BLOCK
403
LOAD F
LOAD G
BASIC
BLOCK
404
LOAD H
LOAD I
LOAD J
LOAD K
BASIC
BLOCK
405

FIG. 4

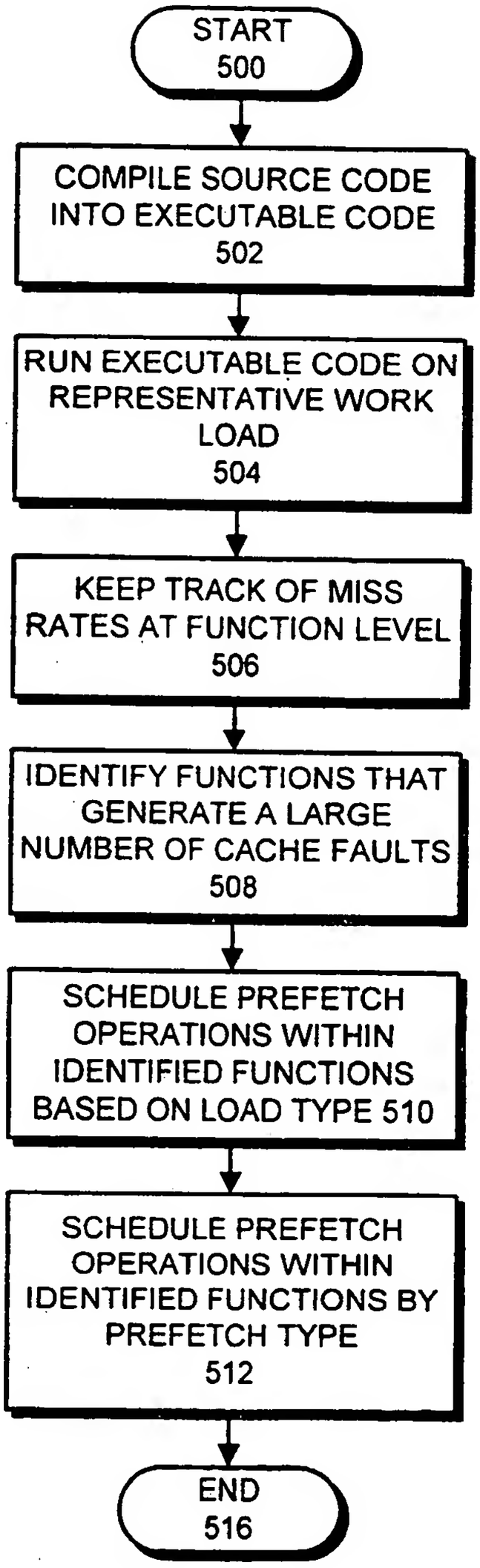

START
500
COMPILE SOURCE CODE INTO EXECUTABLE CODE
502
RUN EXECUTABLE CODE ON REPRESENTATIVE WORK LOAD
504
KEEP TRACK OF MISS RATES AT FUNCTION LEVEL
506
IDENTIFY FUNCTIONS THAT GENERATE A LARGE NUMBER OF CACHE FAULTS
508
SCHEDULE PREFETCH OPERATIONS WITHIN IDENTIFIED FUNCTIONS BASED ON LOAD TYPE 510
SCHEDULE PREFETCH OPERATIONS WITHIN IDENTIFIED FUNCTIONS BY PREFETCH TYPE
512
END
516

FIG. 5

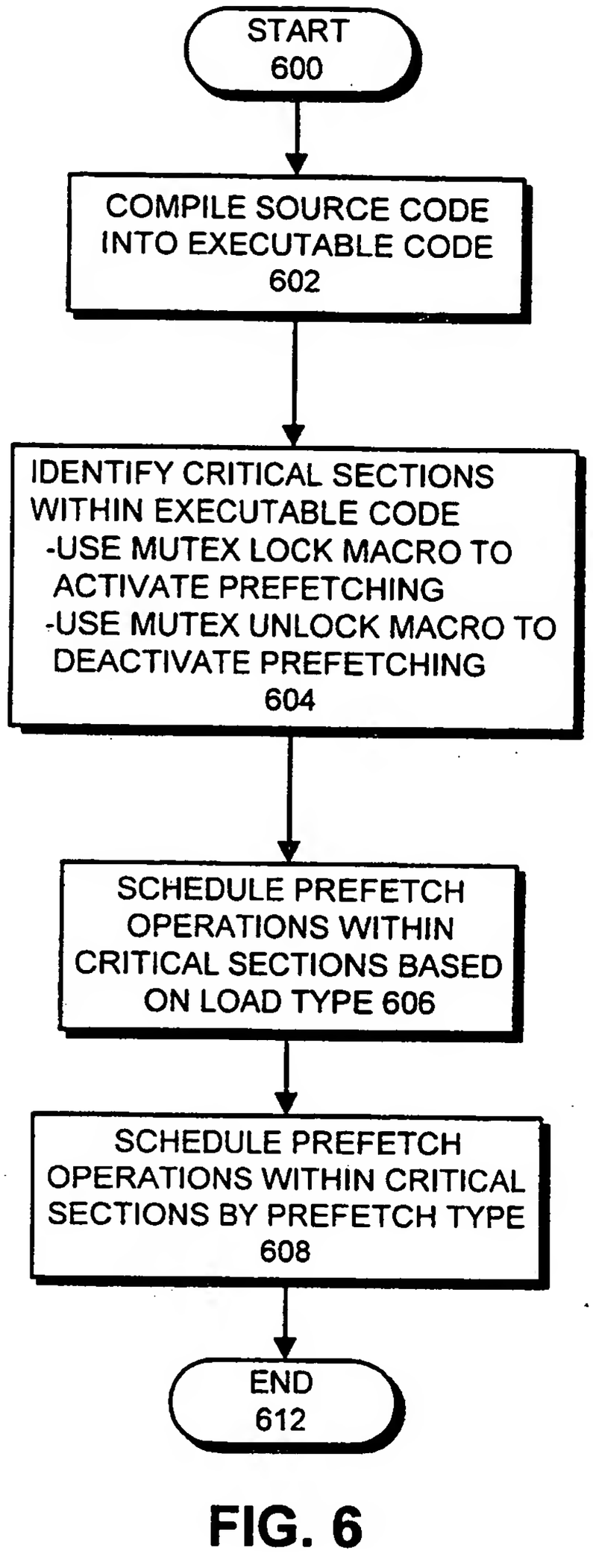

START
600
COMPILE SOURCE CODE INTO EXECUTABLE CODE
602
IDENTIFY CRITICAL SECTIONS WITHIN EXECUTABLE CODE
-USE MUTEX LOCK MACRO TO ACTIVATE PREFETCHING
-USE MUTEX UNLOCK MACRO TO DEACTIVATE PREFETCHING
604
SCHEDULE PREFETCH OPERATIONS WITHIN CRITICAL SECTIONS BASED ON LOAD TYPE 606
SCHEDULE PREFETCH OPERATIONS WITHIN CRITICAL SECTIONS BY PREFETCH TYPE
608
END
612

FIG. 6

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
21 June 2001 (21.06.2001)

PCT

(10) International Publication Number
WO 01/44927 A3

(51) International Patent Classification[7]: G06F 9/45, 9/38

(21) International Application Number: PCT/US00/41668

(22) International Filing Date: 27 October 2000 (27.10.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/434,715 5 November 1999 (05.11.1999) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventors: KOSCHE, Nicolai; 725 Fell Street #9, San Francisco, CA 94117 (US). DAMRON, Peter; 4430 Romilly Way, Fremont, CA 94536 (US).

(74) Agent: PARK, Richard; 508 2nd Street, Suite 201, Davis, CA 95616 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(88) Date of publication of the international search report:
10 May 2002

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR PERFORMING PREFETCHING AT THE FUNCTION LEVEL

(57) Abstract: One embodiment of the present invention provides a system for compiling source code into executable code that performs prefetching for memory operations within regions of code that tend to generate cache misses. The system operates by compiling a source code module containing programming language instructions into an executable code module containing instructions suitable for execution by a processor. Next, the system runs the executable code module in a training mode on a representative workload and keeps statistics on cache miss rates for functions within the executable code module. These statistics are used to identify a set of "hot" functions that generate a large number of cache misses. Next, explicit prefetch instructions are scheduled in advance of memory operations within the set of hot functions. In one embodiment, explicit prefetch operations are scheduled into the executable code module by activating prefetch generation at a start of an identified function, and by deactivating prefetch generation at a return from the identified function. In embodiment, the system further schedules prefetch operations for the memory operations by identifying a subset of memory operations of a particular type within the set of hot functions, and scheduling explicit prefetch operations for memory operations belonging to the subset.

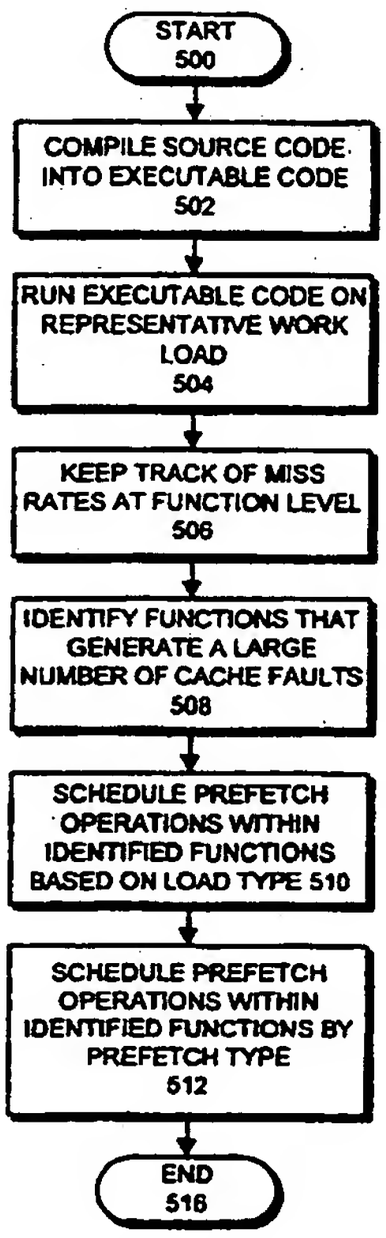


FIG. 5

WO 01/44927 A3

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Interr onal Application No
PCT/US 00/41668

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/45 G06F9/38

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 930 507 A (HAYASHI MASAKAZU ET AL) 27 July 1999 (1999-07-27) the whole document ---	1-31
X	EP 0 883 059 A (NIPPON ELECTRIC CO) 9 December 1998 (1998-12-09) the whole document ---	1-31
X A	US 5 862 385 A (IITSUKA TAKAYOSHI) 19 January 1999 (1999-01-19) abstract column 1, line 1 -column 6, line 28 --- -/--	1,9,17, 25 2-8, 10-16, 18-24, 26-31

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
7 February 2002	15/02/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx 31 651 epo nl, Fax: (+31-70) 340-3016	Beltrán-Escavy, J

INTERNATIONAL SEARCH REPORT

Inter ·onal Application No
PCT/US 00/41668

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category ˜	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
X A	US 5 854 934 A (HSU WEI ET AL) 29 December 1998 (1998-12-29) abstract column 1, line 1 -column 6, line 55	1,9,17, 25 2-8, 10-16, 18-24, 26-31
X A	EP 0 743 598 A (HEWLETT PACKARD CO) 20 November 1996 (1996-11-20) abstract page 1, line 1 -page 2, line 57	1,9,17, 25 2-8, 10-16, 18-24, 26-31
A	US 5 964 867 A (ANDERSON JENNIFER-ANN M ET AL) 12 October 1999 (1999-10-12) abstract column 1, line 1 -column 7, line 17	1-31
A	LUK C-K ET AL: "COMPILER-BASED PREFETCHING FOR RECURSIVE DATA STRUCTURES" ACM SIGPLAN NOTICES, ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, US, vol. 31, no. 9, 1 September 1996 (1996-09-01), pages 222-233, XP000639234 ISSN: 0362-1340 the whole document	1-31
A	CHI-HUNG CHI: "COMPILER'S NEW ROLE IN DATA CACHE PREFETCHING" TECHNOLOGY AND FOUNDATIONS. INFORMATION PROCESSING '94. HAMBURG, AUG. 28 - SEPT. 2, 1994, PROCEEDINGS OF THE IFIP WORLD COMPUTER CONFERENCE, AMSTERDAM, NORTH HOLLAND, NL, vol. 1 CONGRESS 13, 28 August 1994 (1994-08-28), pages 189-194, XP000478803 the whole document	1-31

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Interr ›nal Application No
PCT/US 00/41668

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5930507 A	27-07-1999	JP 8328870 A	13-12-1996
EP 0883059 A	09-12-1998	JP 3156761 B2	16-04-2001
		JP 10333916 A	18-12-1998
		EP 0883059 A2	09-12-1998
US 5862385 A	19-01-1999	JP 7084799 A	31-03-1995
US 5854934 A	29-12-1998	NONE	
EP 0743598 A	20-11-1996	US 5704053 A	30-12-1997
		DE 69615445 D1	31-10-2001
		EP 0743598 A2	20-11-1996
US 5964867 A	12-10-1999	NONE	